Low Time Complexity Algorithms for Path Computation in Cayley Graphs

Daniela Aguirre-Guerrero, Guillaume Ducoffe, Lluis Fabrega, Pere Vila, David Coudert

Low Time Complexity Algorithms for Path Computation in Cayley Graphs

D. Aguirre-Guerrero[1,2], G. Ducoffe[3,4], L. Fàbrega[1], P. Vilà[1], and D. Coudert[5]

[1]Universitat de Girona, Institute of Informatics and Applications, Girona 17003, Spain
[2]Universidad Autónoma Metropolitana, Department of Communications and Information Systems, Lerma 52005, Mexico
[3]National Institute for Research and Development in Informatics, Bucharest 011455, Romania
[4]University of Bucharest, Research Institute (ICUB) and Faculty of Mathematics and Computer Science, Bucharest 011455, Romania
[5]Université Côte d'Azur, Inria, CNRS, I3S, France

Abstract

We study the problem of path computation in Cayley Graphs (CG) from an approach of word processing in groups. This approach consists in encoding the topological structure of CG in an automaton called *Diff*, then techniques of word processing are applied for computing the shortest paths. We present algorithms for computing the K-shortest paths, the shortest disjoint paths and the shortest path avoiding a set of nodes and edges. For any CG with diameter D, the time complexity of the proposed algorithms is $O(KD|Diff|)$, where $|Diff|$ denotes the size of *Diff*. We show that our proposal outperforms the state of art of topology-agnostic algorithms for disjoint shortest paths and stays competitive with respect to proposals for specific families of CG. Therefore, the proposed algorithms set a base in the design of adaptive and low-complexity routing schemes for networks whose interconnections are defined by CG.

Keywords: Cayley graphs; path computation; K-shortest paths; interconnection networks

1 Introduction

Let $G = \langle S|R\rangle$ be a finitely presented group, where S and R are the set of generators and relators, respectively [13, Section 2.2]. The *Cayley Graph* (CG) of G with respect to S is denoted by $\Gamma(G, S)$. The set of vertices is given by the set of group elements. Let $g, h \in G$, there is an edge from g to h if and only if $g \cdot s = h$ for some $s \in S \cup S^{-1}$, where S^{-1} is the set of inverses of G. Cayley graphs are vertex-transitive graphs with degree $\Delta = |S \cup S^{-1}|$. In the remainder of this paper, the elements of G and vertices in $\Gamma(G, S)$ are used interchangeably.

Since S. Akers introduced a group theoretic model for interconnection networks [2], CG have been used as model of a wide variety of communication networks [6, 7, 10, 19, 20]. Traditional routing schemes such as the *Valiant Routing* algorithm [17] and the *Universal Globally-Adaptive Load-balanced* algorithm [15] have been proposed to work on these topologies. These schemes apply topology-agnostic algorithms of K-shortest paths [4] and K-shortest disjoint paths [18] which have high memory and time requirements.

However, routing schemes with low space and time complexity can be designed taking advantage of the vertex-transitive property of CG [9]. Recent proposals in this direction include

algorithms for path computation in specific families of CG [12,14]. To the best of our knowledge, there is no topology-agnostic algorithms for path computation in CG.

To overcome this limitation, this paper presents low time complexity algorithms for path computation in CG. These algorithms extend the work presented in [1], where the topological structure of CG is encoded in a Deterministic Finite Automaton (DFA) called *Diff*. Then, techniques of word processing are applied to design a simple shortest path routing scheme for CG. We use the same approach to propose algorithms for computing the K-shortest paths, the shortest disjoint paths and the shortest path avoiding a set of nodes and edges.

For a CG with diameter D, the time complexity of the proposed algorithms is $O(KD|Diff|)$. We show that our algorithm for disjoint paths outperforms the topology-agnostic solution [4]. Moreover, we show that for many families of CG used as model of communication networks, our algorithm for K-shortest disjoint paths stays competitive with respect to the algorithms presented in [12, 18].

The remainder of the paper is as follows. Section 2 surveys the recent proposals of path computation in CG. Section 3 introduces the theoretical background about word processing in groups. Sections 4, 5 and 6 present, respectively, the algorithms for computing the K-shortest paths, the shortest disjoint paths and the shortest path avoiding a set of nodes and edges. Finally, Section 7 gives the conclusions.

2 Related work

Most of the routing schemes proposed for CG based networks apply topology-agnostic algorithms for path computation. The best-know algorithms for computing the K-shortest paths and K-disjoint paths in a graph with n vertices and m edges, run in time, respectively, $O(n+m+K)$ and $O(Knm)$ [4, 18]. In order to reduce the time and space complexity, path computation algorithms have been proposed for specific families of CG.

A routing algorithm for Boreal CG, which is fault-tolerant but is not shortest path and does not ensure packet delivery is proposed in [14]. A shortest path algorithm for pancake graphs is presented in [16]. This algorithm runs in time $O(K\Delta^3)$ if the computed paths have length less than $2\Delta + 16$. An algorithm for computing the K-shortest vertex-disjoint paths in CG of abelian groups, which includes several families of CG such as hypercubes and buble-sort graphs, is presented in [12]. This algorithm has time complexity in $O(K\Delta D)$. Finally, [1] presents a shortest path algorithm for CG based data center networks with time complexity $O(D^2)$.

The algorithms proposed in this paper extend the work of [1]. In contrast to the algorithms presented in [12, 14, 16], our algorithms are topology-agnostic and have no constraints on the size of the computed paths. In addition, we present an algorithm for computing the shortest path avoiding a set of nodes and edges, which is useful in the design of fault-tolerant routing schemes.

3 Word processing in Cayley graphs

3.1 Words as Paths and Nodes

Before explaining how paths and nodes in a CG can be represented through words, it is necessary to introduce some definitions and notation from geometric group theory. Let A be an alphabet, such that there is a bijective map

$$\phi : S \cup S^{-1} \rightarrow A, \tag{1}$$

where S and S^{-1} are the sets of generators and inverses of G, respectively. Eq. (1) assigns each generator and its inverse to lowercase and uppercase variants of the same letter. A letter $X \in A$

is said to be *the inverse of a letter* $x \in A$ if and only if $X = \phi(s^{-1})$ and $x = \phi(s)$, where s^{-1} is the inverse of $s \in S \cup S^{-1}$.

Let w be a word over A. We define the following words:

- The *inverse of* w, denoted by w^{-1}, is given by the reverse string of the inverse letters of w.
- A *substring of* w, denoted by $w(i)$, is given by the first i letters in w. If $i > |w|$, $w(i) = w$.
- The *reduced form of* w, denoted by w_{red}, results from removing substrings of the form uu^{-1} from w.

Definition 1. *Let $F(A)$ be the free group over A, which consists of all reduced words over A including the symbol e_A that denotes the null string. Then, there is a group homomorphism $\gamma : F(A) \to G$ that assigns a unique set of words in $F(A)$ to each group element in G. The symbol e_A is assigned to the identity element, i.e. Id, [5, Definition 2.1.8]. We define an equivalence relation on $F(A)$, denoted by $=_G$, such that $w =_G v$ if and only if w and v represent the same group element under γ. The set of words representing the same group element is defined by the equivalence class $[w]$.*

Turning now to the CG of G, i.e. $\Gamma(G, S)$, let us assume that each edge is labelled according to Eq. (1), see Fig. 1a. Thereby paths in $\Gamma(G, S)$ can be represented by words in $F(A)$, where the symbol e_A represents the empty path. In Fig. 1a, the word *abc* represents a path between nodes (1234) and (2341). Moreover, words in the same equivalence class $[w]$ represent paths between the same pair of nodes. In Fig. 1a, the words *abc* and *babca* are in the same equivalence class as they represent the same permutation and thus the same group element (2341).

Let $w \in [w]$, such that $\gamma(w) = g$. Hereafter $\overline{w}$ denotes the node g, whereas $\widehat{w}$ denotes a path represented by the word w_{red}, see Fig. 1b. Since $\widehat{w}$ represents different paths, to refer a specific path in this notation it is necessary to indicate the first node in the path. Then, $\widehat{w}(\overline{w_1})$ denotes a path starting at node $\overline{w_1}$ and given by a list of nodes $[\overline{w_1}, \ldots, \overline{w_l}]$. In Fig. 1b, the path $\widehat{abc}(\overline{b})$ is given by the list of nodes $[\overline{b}, \overline{ba}, \overline{aba}, \overline{abac}]$. In particular, $\widehat{w}(\overline{e_A})$ represents a path from $\overline{e_A}$ to $\overline{w}$. In Fig. 1b, the path $\widehat{abc}(\overline{e_A})$ represents a path from $\overline{e_A}$ to $\overline{abc}$. To give a unique representation in $F(A)$ for nodes and paths in $\Gamma(G, S)$, we need to define a canonical form over the set of equivalence classes of Definition 1.

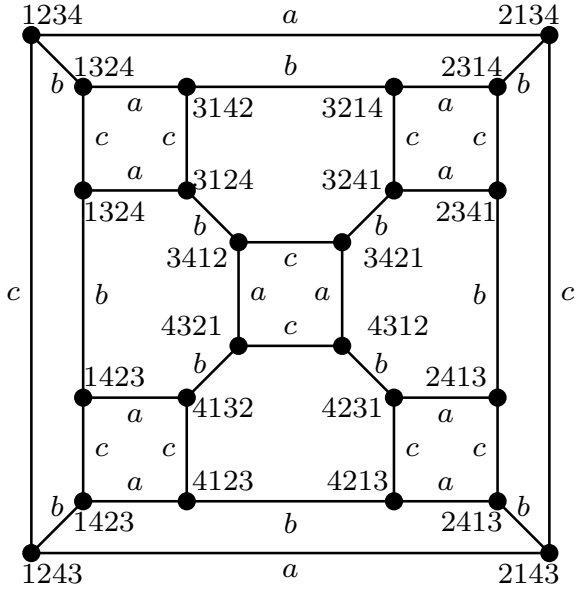
Definition 2. *Let $<_A$ be a lexicographical order over A. Let $w, v \in F(A)$, we say that w is ShortLex than v, if w is shorter than v, i.e. $|w| < |v|$, or w and v have the same length but w comes before v in the order $<_A$. We define the language of the ShortLex words in $F(A)$ representing a unique group element in G as*

$$L = \{w \in F(A) : w \leq_A v, \forall v \in F(A) \text{ s.t. } w =_G v\}. \tag{2}$$

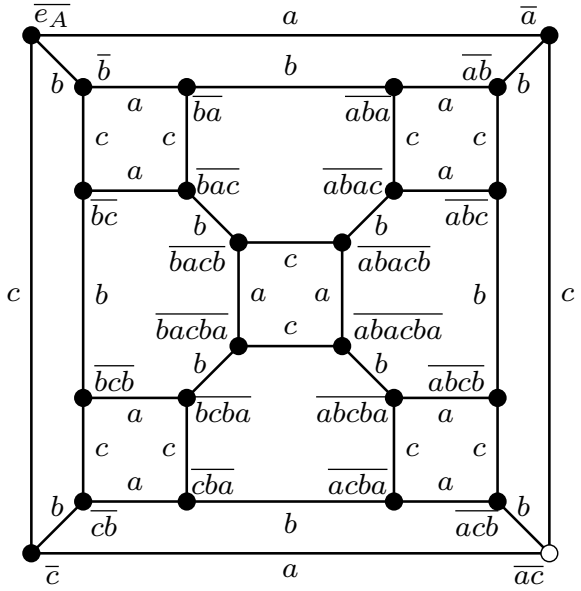
Therefore there is a bijective map $\pi : G \to L$ that assigns each group element in G to its *ShortLex* representative word in $F(A)$ [9, Definition 13.6]. Language L defines a canonical form for the set of equivalence classes of Definition 1. Hence words in L give a unique representation for nodes in $\Gamma(G, S)$, and the shortest paths between any pair of nodes in $\Gamma(G, S)$. From now on, we refer to these paths as *ShortLex* paths.

3.2 Computing the Shortest Path

From the previous subsection, there is a path between any two nodes $\overline{u}$ to $\overline{v}$ given by the word $u^{-1}v$. The path $\widehat{u^{-1}v}$ goes from $\overline{u}$ to $\overline{e_A}$ and from $\overline{e_A}$ to $\overline{v}$. Then, the problem of computing the



(a)



(b)

Figure 1: Cayley graph of the symmetric group $Sym(4)$ generated by the permutations $S = \{(2,1,3,4),(1,3,2,4),(1,2,4,3)\}$, where $S = S^{-1}$. **(a)** Each generator is assigned to a letter in the alphabet $A = \{a,b,c\}$ as follows: $(2,1,3,4) \to a$, $(1,3,2,4) \to b$ and $(1,2,4,3) \to c$. **(b)** Nodes are labelled with a word representing the shortest path from node $\overline{e_A}$ to each of them.

shortest path between $\overline{u}$ and $\overline{v}$ is equivalent to computing the canonical form of the word $u^{-1}v$, i.e. $w \in L$ such that $w =_G u^{-1}v$. This problem is called the Minimum Word Problem (MWP) and can be resolved in time $O(|u^{-1}v|^2)$ for *ShortLex Automatic Groups*[1] (SAG) [5, Theorem 2.3.10].

Lemma 1. *We say that $G = \langle S|R\rangle$ is a ShortLex Automatic Group (SAG), if G satisfies the k-fellow-traveler property, which states that there exists a constant k (depending on G), such that for all ShortLex paths $\widehat{u}(\overline{u_1}) = [\overline{u_1}, \ldots, \overline{u_l}]$ and $\widehat{v}(\overline{v_1}) = [\overline{v_1}, \ldots, \overline{v_{l-1}}]$ in $\Gamma(G, S)$ beginning at the same node, i.e. $\overline{u_1} = \overline{v_1}$, and whose end nodes are adjacent, the uniform distance between $\widehat{u}$ and $\widehat{v}$ is bounded by k, i.e. $d(\overline{u_i}, \overline{v_i}) \leq k$, where $i \in [1, l-1]$, and thus $d(\widehat{u}, \widehat{v}) \leq k$ [5, Lemma 2.3.2].*

The process of computing the canonical form of $u^{-1}v$ consists in applying a set of rewriting rules to $u^{-1}v$. For CG arising from SAG, these rules can be encoded in DFA called *Word-Difference* Automaton (*Diff*), which is used in the algorithms presented in this paper. Given a group presentation of a SAG, i.e. $G = \langle S|R\rangle$ and an alphabet A satisfying Eq. (1), the Knuth-Bendix completion algorithm [11] is able to compute *Diff*[2] and others DFA such as the *word-acceptor* that recognizes the language defined by Eq. (2). Let us now explain what is *Diff* and how it is used for computing the canonical form of a word.

Definition 3. *The word-difference automaton of G, i.e. $Diff = (WD, B, \delta, e_A)$, is a DFA consisting of: a set of states given by $WD = \{w \in L : |w| \leq k\}$, where e_A is the start state; an alphabet $B = A \cup \{e_A\} \times A \cup \{e_A\}$; and a transition function $\delta : WD \times B \to WD$. This automaton accepts a tuple of words (r, t), where $|r| = |t|$, if and only if $r \leq_A t$ and the canonical form of any word $r(i)^{-1}t(i)$ is in WD. The state after reading a tuple of substrings $(r(i), t(i))$, i.e. $q^{(r(i),t(i))}$, is given by the canonical form of $r(i)^{-1}t(i)$ [9, Section 13.2.2].*

Corollary 1. *If Diff accepts (r, t), then paths $\widehat{r}$ and $\widehat{t}$, beginning at the same node, satisfy the following conditions: 1) $d(\widehat{r}, \widehat{t}) \leq k$; 2) $|\widehat{r}| \leq |\widehat{t}|$ due to r could be not in canonical form; and 3) the ShortLex path from $\overline{r}$ to $\overline{t}$ is $\widehat{q^{(r,t)}}$, see Fig. 2. Therefore, if $q^{(r,t)} = e_A$, $\widehat{r}$ and $\widehat{t}$ join the same pair of nodes, i.e. $r =_G t$; and if $q^{(r,t)} \neq e_A$, $\widehat{r}$ and $t(\widehat{q^{(r,t)}})^{-1}$ join the same pair of nodes, i.e. $r =_G t(q^{(r,t)})^{-1}$.*

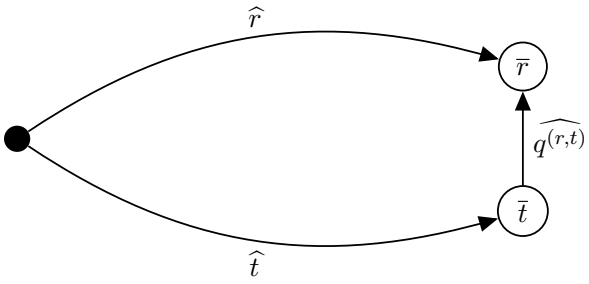


Figure 2: Geometric representation of a tuple of words (r, t) accepted by the automaton *Diff*, such that the final state is $q^{(r,t)}$. The *ShortLex* path between the end nodes of paths $\widehat{r}$ and $\widehat{t}$, i.e. $\overline{r}$ and $\overline{t}$, is given by $\widehat{q^{(r,t)}}$.

The computation of the shortest path between any pair of nodes $\overline{u}$ and $\overline{v}$ in $\Gamma(G, S)$ consists in searching in $w = u^{-1}v$ for the shortest substring $w(i) \notin L$. After finding $w(i)$, we search for

[1] It includes all finite groups and several infinite groups.

[2] The software package *KBMAG* [8] implements the Knuth-Bendix completion algorithm. Details of the time requirements can be found in [9, Section 13.3.6].

Table 1: Notation

Parameter	Definition
$\Gamma(G,S)$	Cayley graph of $G = \langle S \vert R \rangle$
$\Delta = \vert S \cup S^{-1} \vert$	Degree of $\Gamma(G,S)$
D	Diameter of $\Gamma(G,S)$
k	*Fellow-traveler* constant of $\Gamma(G,S)$
K	A positive integer
$\overline{u}$, $\overline{v}$	Two nodes in $\Gamma(G,S)$
$\widehat{w}$	The *shortLex* path between $\overline{u}$ and $\overline{v}$

the word $r \in L$, such that $q^{(r,w(i))} = e_A$ [9, Section 13.1.7]. Then $w(i)$ is replaced by r_{red} in w. This process is repeated until the canonical form of w is found. The resulting word represents the *ShortLex* path between $\overline{u}$ and $\overline{v}$ [1, Algorithm 4].

The algorithms presented in the following sections assume that nodes are labeled with their *ShortLex* representative word in L according to Definition 2. Algorithm 1 of [1] presents the steps to perform this label assignment. Let $u, v, w \in L$, the notations used in the following sections are presented in Table 1.

4 Computing the K-shortest paths

In this section, recursive algorithms for computing (in increasing order) the K-shortest paths between any $\overline{u}$ and $\overline{v}$ are presented. This process is equivalent to compute the K-*ShortLex* words $p \in F(A)$, such that $p =_G w$.

Lemma 2. *Let $\mathcal{P}_j(w) \subset F(A)$ be the set of words representing the shortest paths between $\overline{u}$ and $\overline{v}$ that are at distance at most jk from $\widehat{w}$, i.e. $\mathcal{P}_j(w) = \{t \in F(A) : t =_G w, |t| = |w|$ and $d(\widehat{t}, \widehat{w}) \leq jk\}$. Then, $\mathcal{P}_j(w)$ can be built recursively as follows:*

$$\mathcal{P}_j(w) = \{t \in F(A) : \exists r \in \mathcal{P}_{j-1}(w) \text{ s.t. } q^{(r,t)} = e_A\}, \tag{3}$$

where $j > 0$ and $\mathcal{P}_0(w) = \{w\}$.

Proof. Clearly, two paths are at distance 0 if and only if they are the same path. So, $\mathcal{P}_0(w) = \{w\}$. We now proceed by induction. Assuming Eq. (3) holds for $\mathcal{P}_{j-1}(w)$, we will prove it for $\mathcal{P}_j(w)$. Eq. (3) implies that: 1) $t =_G w$ due to $t =_G r$ (by Corollary 1) and $r =_G w$ (by the induction hypothesis); 2) $|t| = |w|$ due to $|t| = |r|$ (by Definition 3) and $|r| = |w|$ (by the induction hypothesis); and 3) by the triangle inequality $d(\widehat{t}, \widehat{w}) \leq jk$ due to $d(\widehat{r}, \widehat{t}) \leq k$ (by Corollary 1) and $d(\widehat{r}, \widehat{w}) \leq (j-1)k$ (by the induction hypothesis). Therefore Eq. (3) defines the shortest paths between $\overline{u}$ and $\overline{v}$ that are at distance at most jk from $\widehat{w}$ and Lemma 2 holds for $j \geq 0$. □

Given the set $\mathcal{P}_{j-1}(w)$, Algorithm 1 computes $\mathcal{P}_j(w)$. The algorithm implements Eq. (3) as follows: for each word $r \in \mathcal{P}_{j-1}(w)$, the set of words $t \in F(A)$ such that $q^{(r,t)} = e_A$, i.e. $\mathcal{U}_r$, is computed (line 3). The set $\mathcal{P}_j(w)$ results from the union of all $\mathcal{U}_r$ (line 4).

Lemma 3. *Algorithm 1 computes $\mathcal{P}_j(w)$ in time $O(|\mathcal{P}_{j-1}(w)||w||Diff|)$.*

Proof. The set $\mathcal{U}_r$ is computed $|\mathcal{P}_{j-1}(w)|$ times. The computation of $\mathcal{U}_r$ can be done by traversing each state of the automaton *Diff* at most $|r| = |w|$ times [9, Section 13.1.7]. Therefore, Algorithm 1 computes $\mathcal{P}_j(w)$ in time $O(|\mathcal{P}_{j-1}(w)||w||Diff|)$. □

Corollary 2. *Words in $\mathcal{P}_{\lceil D/k \rceil}(w)$ represent all the shortest paths between $\overline{u}$ and $\overline{v}$ due to $d(\widehat{w}, \widehat{p}) \leq D$ for any path $\widehat{p}$ between $\overline{u}$ and $\overline{v}$.*

Algorithm 1 Compute the set of paths $\mathcal{P}_j(w)$.

Input: The automaton $Diff = (WD, B, \delta, e_A)$.
Input: The set of words $\mathcal{P}_{j-1}(w) \subset F(A)$.
Output: The set of words $\mathcal{P}_j(w) \subset F(A)$.

$\mathcal{P}_j \leftarrow \mathcal{P}_{j-1}$
for $r \in \mathcal{P}_{j-1}$ **do**
  $\mathcal{U}_r \leftarrow \{t \in F(A)\colon q^{(r,t)} = e_A\}$
  $\mathcal{P}_j \leftarrow \mathcal{P}_j \cup \mathcal{U}_r$
return $\mathcal{P}_j$

Algorithm 2 Compute the set of paths $[w]_i$.

Input: The automaton $Diff = (WD, B, \delta, e_A)$.
Input: The set of words $[w]_{i-1} \subset F(A)$.
Output: The set of words $[w]_i \subset F(A)$.

$[w]_i \leftarrow [w]_{i-1}$
$\mathcal{R} \leftarrow \{r \in [w]_{i-1} : |r| > |w| + (i-2)k\}$
for $r \in \mathcal{R}$ **do**
  $\mathcal{V}_r \leftarrow \{p \in F(A) : p = (t(q^{(r,t)})^{-1})_{red}$ and $\exists t \in F(A)$ s.t. $q^{(r,t)} \in WD\}$
  $[w]_i \leftarrow [w]_i \cup \mathcal{V}_r$
return $[w]_i$

Theorem 1. *Let $[w]_i \subset F(A)$ be the set of words representing the paths between $\overline{u}$ and $\overline{v}$ whose length is at most $|w| + ik$ for $i \geq 0$, i.e. $[w]_i = \{p \in F(A) : p =_G w$ and $|\widehat{p}| \leq |\widehat{w}| + ik\}$. Then, $[w]_i$ can be built recursively as follows:*

$$\begin{aligned}[w]_i = \{&p \in F(A) : p = (t(q^{(r,t)})^{-1})_{red},\\ &\exists r \in [w]_{i-1} \text{ and } t \in F(A) \text{ s.t. } q^{(r,t)} \in WD\},\end{aligned} \tag{4}$$

where $i > 0$ and $[w]_0 = \mathcal{P}_{\lceil D/k \rceil}(w)$.

Proof. We proceed by induction. By Corollary 2, $[w]_0 = \mathcal{P}_{\lceil D/k \rceil}(w)$. Assuming Eq. (4) to hold for $[w]_{i-1}$, we will prove it for $[w]_i$. Eq. (4) implies that: 1) $p =_G w$ due to $p =_G r$ (by Corollary 1) and $r =_G w$ (by the induction hypothesis); and 2) $|\widehat{p}| \leq |\widehat{w}| + ik$ due to $|p| = |t| + |q^{(u,v)}|$, where $|q^{(u,v)}| \leq k$ (by Definition 3), $|t| = |r|$ (by Definition 3) and $|r| \leq |w| + (i-1)k$ (by the induction hypothesis). Therefore, Eq. (4) defines the set of paths $\widehat{p}$ between $\overline{u}$ and $\overline{v}$ that satisfy $|\widehat{p}| \leq |\widehat{w}| + ik$ and Theorem 1 holds for $i \geq 0$. □

Corollary 3. *Let i be the smallest positive integer such that $[w]_i = [w]_{i-1}$. Then, words in $[w]_{i-1}$ represent all paths between $\overline{u}$ and $\overline{v}$.*

Corollary 4. *The set of words p representing the paths between $\overline{u}$ and $\overline{v}$ such that $|\widehat{w}| + (i-1)k < |\widehat{p}| \leq |\widehat{w}| + ik$ is given by:*

$$\begin{aligned}[w]_i \setminus [w]_{i-1} = \{&p \in F(A) \setminus [w]_{i-1} : p = (t(q^{(r,t)})^{-1})_{red},\\ &\exists\, r \in [w]_{i-1} \setminus [w]_{i-2} \text{ and } t \in F(A) \text{ s.t. } q^{(r,t)} \in WD\}.\end{aligned} \tag{5}$$

Given the set $[w]_{i-1}$, Algorithm 2 computes $[w]_i$. First, $[w]_i$ is initialized to $[w]_{i-1}$ (line 1). Then, $[w]_{i-1} \setminus [w]_{i-2}$ is computed and stored in $\mathcal{R}$ (line 2). Finally, Eq. (5) is implemented (*for* loop at line 3). The set $[w]_i$ results from the union of $[w]_{i-1}$ and $[w]_i \setminus [w]_{i-1}$.

Lemma 4. *Algorithm 2 computes* $[w]_i$ *in time* $O(|[w]_{i-1} \setminus [w]_{i-2}|D|Diff|)$.

Proof. The set $\mathcal{V}_r$ is computed $|[w]_{i-1} \setminus [w]_{i-2}|$ times. The computation of $\mathcal{V}_r$ can be done by traversing each state of the automaton *Diff* at most $|r| \leq D$ times [9, Section 13.1.7]. Therefore, Algorithm 2 computes $[w]_i$ in time $O(|[w]_{i-1} \setminus [w]_{i-2}|D|Diff|)$. □

Corollary 5. *Algorithm 2 computes the* K*-shortest paths in time* $O(KD|Diff|)$.

5 Computing the shortest disjoint paths

Let $\mathcal{D}_E(u, v) \subset L$ be the set of words representing the shortest edge-disjoint paths between $\overline{u}$ and $\overline{v}$. Algorithm 3 presents the steps for computing $\mathcal{D}_E(u, v)$, where $|\mathcal{D}_E(u, v)| = \Delta$ since CG are vertex-transitive [3]. First, $\mathcal{D}_E(u, v)$ is initialized to w (lines 1-2). The strategy to identify edge-disjoint paths is to record the edges of paths in $\mathcal{D}_E(u, v)$, which is given by the set E_D (line 3). The shortest edge-disjoint path from paths in $\mathcal{D}_E(u, v)$ is searched in $[w]_i \setminus [w]_{i-1}$. For each $z \in [w]_i \setminus [w]_{i-1}$, the intermediate edges in the path $\widehat{z}(\overline{u})$ are computed, i.e. $\mathcal{E}_z$ (line 6). If $E_D \cap \mathcal{E}_z = \emptyset$, then $\widehat{z}$ is edge-disjoint from paths in $\mathcal{D}_E(u, v)$ (*if* condition at line 7). Thereby z is added to $\mathcal{D}_E(u, v)$ and E_D is updated (lines 8-9). In addition, it is checked if all edge-disjoint paths have been computed, if so $\mathcal{D}_E(u, v)$ is returned (*if* condition at line 10). Otherwise, this process is repeated for $i + 1$, and so on until $|\mathcal{D}_E(u, v)| = \Delta$ (*while* loop at line 4).

Algorithm 3 Compute the shortest edge-disjoint paths.

Input: Two words $u, v \in L$ representing two nodes.

Output: A set of words $\mathcal{D}_E(u, v) \subset F(A)$ representing the Δ-shortest edge-disjoint paths from $\overline{u}$ to $\overline{v}$.

1: $w \leftarrow$ the canonical form of $u^{-1}v$
2: $i \leftarrow 1$, $[w]_{i-1} \leftarrow \emptyset$, $[w]_i \leftarrow \{w\}$, $\mathcal{D}_E \leftarrow \{w\}$
3: $E_D \leftarrow \{(e, f) \in L \times L : e =_G uw(i)$ and $f =_G uw(i + 1)$, for $0 < i < |w|\}$
4: **while** $[w]_i \neq [w]_{i-1}$ **do**
5: **for** $z \in [w]_i \setminus [w]_{i-1}$ **do**
6: $\mathcal{E}_z = \{(e, f) \in L \times L : e =_G uz(i)$ and $f =_G uz(i + 1)$, for $0 < i < |z|\}$
7: **if** $E_D \cap \mathcal{E}_z = \emptyset$ **then**
8: Add z to $\mathcal{D}_E$
9: $E_D \leftarrow E_D \cup \mathcal{E}_z$
10: **if** $|\mathcal{D}_E| = \Delta$ **then**
11: **return** $\mathcal{D}_E$
12: $[w]_{i-1} \leftarrow [w]_i$, $i \leftarrow i + 1$
13: Compute $[w]_i$
14: **return** $\mathcal{D}_E$

Similarly, the set of shortest node-disjoint paths, i.e. $\mathcal{D}_V(u, v)$, can be computed. The record of the intermediate nodes of paths in $\mathcal{D}_V(u, v)$ is kept in a set V_D. Then, the shortest node-disjoint path from paths in $\mathcal{D}_V(u, v)$ is searched in $[w]_i \setminus [w]_{i-1}$. A path whose intermediate nodes are not in V_D is node-disjoint from paths in $\mathcal{D}_V(u, v)$. In CG that are not edge-transitive,

the number of node-disjoint paths could be less than Δ [3]. In this case, the algorithm will finish when all paths between $\overline{u}$ and $\overline{v}$ have been computed. This condition is satisfied when $[w]_i = [w]_{i-1}$ due to Corollary 3 (*while* loop at line 4).

Lemma 5. *Algorithm 3 computes $\mathcal{D}_E(u, v)$, in time $O(|[w]_{i-1}||w||Diff|)$.*

Proof. The set $[w]_i$ is computed once. Therefore, Algorithm 3 finishes in time $O(|[w]_{i-1}||w||Diff|)$ due to Lemma 4. □

Corollary 6. *If the largest path in $\mathcal{D}_E(u, v)$ is the K-th shortest path between $\overline{u}$ and $\overline{v}$. Then, Algorithm 3 computes $\mathcal{D}_E(u, v)$ in time $O(KD|Diff|)$ due to Corollary 5.*

6 Computing the shortest path avoiding a set of nodes and edges

Let $V_f \subset L$ representing a set of nodes in $\Gamma(G, S)$, such that $u, v \notin V_f$. Let $z \in F(A)$ representing the shortest path from $\overline{u}$ to $\overline{v}$ avoiding nodes in V_f. Algorithm 4 computes z, if it exists. Otherwise, it returns *Null*. Algorithm 4 starts computing $[w]_i$, for $i = 0$ (lines 1-2). Then, the word avoiding nodes in V_f is searched in $[w]_i$ as follows: for each $z \in [w]_i \setminus [w]_{i-1}$, the intermediate nodes in the path $\widehat{z}(\overline{u})$ are computed, i.e. $\mathcal{V}_z$, (line 5). If $V_f \cap \mathcal{V}_z = \emptyset$, then $\widehat{z}(\overline{u})$ does not contain nodes in V_f and z is returned (*if* condition at line 6). Otherwise, the process is repeated for $i + 1$, and so on until the path is found, if it exists (*while* loop at line 3).

Similarly, the shortest path avoiding a set of edges $E_f \subset L \times L$ can be computed. It is searched in $[w]_i$ as follows: for each $z \in [w]_i \setminus [w]_{i-1}$, the intermediate edges in the path $\widehat{z}(\overline{u})$ are computed, i.e. $\mathcal{E}_z$. If $E_f \cap \mathcal{E}_z = \emptyset$, then $\widehat{z}(\overline{u})$ does not contain edges in E_f.

Lemma 6. *Algorithm 4 runs in time $O(|[w]_{i-1}||w||Diff|)$.*

Proof. The set $[w]_i$ is computed once. Therefore, Algorithm 4 finishes in time $O(|[w]_{i-1}||w||Diff|)$ due to Lemma 4. □

Corollary 7. *If $\widehat{z}$ is the K-th shortest path between $\overline{u}$ and $\overline{v}$. Then, Algorithm 4 computes z in time $O(KD|Diff|)$ due to Corollary 5.*

Algorithm 4 Compute the shortest paths avoiding a set of nodes.

Input: Two words $u, v \in L$ representing two nodes.
Input: A set of words $V_f \subset L$ representing a set of nodes, such that $u, v \notin V_f$.
Output: A word $z \subset F(A)$ representing the shortest path from $\overline{u}$ and $\overline{v}$ that does not contains nodes in V_f. *Null*, if such a word does not exist.

1: $w \leftarrow$ the canonical form of $u^{-1}v$
2: $i \leftarrow 0$, $[w]_{i-1} \leftarrow \emptyset$, $[w]_i \leftarrow \{w\}$
3: **while** $[w]_i \neq [w]_{i-1}$ **do**
4: **for** $z \in [w]_i \setminus [w]_{i-1}$ **do**
5: $\mathcal{V}_z = \{v \in L : v =_G uz(i), 1 < i < |z|\}$
6: **if** $V_f \cap \mathcal{V}_z = \emptyset$ **then**
7: **return** z
8: $[w]_{i-1} \leftarrow [w]_i$, $i \leftarrow i + 1$
9: Compute $[w]_i$
10: **return** *Null*

7 Conclusions

We have presented algorithms for computing the K-shortest paths, the shortest disjoint paths and the shortest path avoiding a set of nodes and edges in CG. The proposed algorithms use a DFA called *Diff*, which encodes the topological structure of CG. Then, techniques of word processing are applied to compute the shortest paths. For a CG with diameter D, our algorithms run in time $O(KD|Diff|)$.

The best-known algorithms for computing the K-shortest paths and the K-shortest disjoint paths in a graph with n vertices and m edges, run in time, respectively, $O(n + m + K)$ and $O(Knm)$ [4,18]. On the other hand, the algorithm for computing the K-shortest disjoint paths in CG of abelian groups runs in time $O(K\Delta D)$ [12].

Since *Diff* has size $O(n+m) = O(\Delta n)$ our algorithm for shortest disjoint paths outperforms the state of the art. Furthermore, our solution for K-shortest disjoint paths stays competitive whenever *Diff* and D are small, say $|Diff| = O(\Delta)$ and $D = \log^{O(1)}(n)$, that happens to be the case for many families of CG used as model of communication networks, such as hypercubes and bubble-sort graphs [1]. Therefore, the proposed algorithms set a base in the design of adaptive and low-complexity routing schemes for networks whose interconnections are defined by CG.

Acknowledgments

We wish to thank the referees for their careful reading of the first version of this manuscript, and their useful comments. This research was partially funded by the Mexican Government (CONACYT-SENER 2015-409697), the Spanish Government (TEC 2015-66412-R, TEC 2015-71932-REDT), the Catalan Government (2017-SGR-1551), the Romanian Government (PN 1819 RESINFO-TD/2018, PN 1819-01-01, CCCDI-UEFISCDI. PN 17PCCDI/2018) and the French Government (ANR-11-LABX-0031-01).

References

[1] D. Aguirre-Guerrero, M. Camelo, L. Fàbrega, and P. Vilà. WMGR: A generic and compact routing scheme for data center networks. *IEEE/ACM Transactions on Networking*, 26(1):356–369, Feb. 2018.

[2] S. B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, Apr. 1989.

[3] A. H. Dekker and B. D. Colbert. Network robustness and graph topology. In *27th Australasian Conference on Computer Science*, volume 26 of *ACSC*, pages 359–368, Darlinghurst, Australia, 2004. Australian Computer Society, Inc.

[4] D. B. A. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[5] D. B. A. Epstein, M. S. Paterson, J. W. Cannon, D. F. Holt, S. V. Levy, and W. P. Thurston. *Word Processing in Groups*. A. K. Peters, Ltd., Natick, MA, USA, 1992.

[6] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: A high performance, server-centric network architecture for modular data centers. In *ACM SIGCOMM Conference on Data Communication*, pages 63–74, New York, NY, USA, 2009. ACM.

[7] M. C. Heydemann. *Cayley graphs and interconnection networks*. Springer Netherlands, Dordrecht, 1997.

[8] D. Holt. KBMAG Package: A Knuth-Bendix on Monoids, and Automatic Groups. https://www.gap-system.org/Packages/kbmag.html, 2017. Accessed: 2018-02-23.

[9] D. F. Holt, B. Eick, and E. A. O'Brien. *Handbook of computational group theory*. Discrete mathematics and its applications. Chapman & Hall/CRC, Boca Raton, 2005.

[10] J. Kim, J. Balfour, and W. J. Dally. Flattened butterfly topology for on-chip networks. *IEEE Computer Architecture Letters*, 6(2):37–40, Feb 2007.

[11] D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning*, Symbolic Computation, pages 342–376. Springer Berlin, 1983.

[12] C. Lai. On the construction of all shortest vertex-disjoint paths in cayley graphs of abelian groups. *Theoretical Computer Science*, 571:10 – 20, 2015.

[13] D. Robinson. *A Course in the Theory of Groups*. Number 80 in Graduate Texts in Mathematics. Springer-Verlag, 1982. Second edition, 1996.

[14] J. Ryu, E. Noel, and K. W. Tang. Fault-tolerant routing on borel cayley graph. In *IEEE International Conference on Communications (ICC)*, pages 2872–2877, June 2012.

[15] A. Singh. *Load-balanced Routing in Interconnection Networks*. PhD thesis, Stanford University - Department of Electrical Engineering, 2005.

[16] M. Tokuda, Y. Hirai, and K. Kaneko. An algorithm for k-pairwise cluster-fault-tolerant disjoint paths in a burnt pancake graph. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 651–655, Dec 2015.

[17] L. G. Valiant. A scheme for fast parallel communication. *SIAM Journal on Computing*, 11(2):350–361, 1982.

[18] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[19] J. Yu, E. Noel, and K. W. Tang. A graph theoretic approach to ultrafast information distribution: Borel cayley graph resizing algorithm. *Computer Communications*, 33(17):2093 – 2104, 2010.

[20] J. X. Zhou, Z. L. Wu, S. C. Yang, and K. W. Yuan. Symmetric property and reliability of balanced hypercube. *IEEE Transactions on Computers*, 64(3):876–881, Mar. 2015.